# 23/24 AN2DL Challenge 2

Giacomo Ballabio - 10769576
Christian Biffi - 10787158
Alberto Cavallotti - 10721275

# Deep Sheets

## 1 Introduction

The objective of this project was to train a neural network for time series forecasting on a concealed test set containing 60 time series. The task involved predicting the next 9 points in the first phase and 18 points in the second phase for each time series.

## 2 Dataset analysis and processing

We were provided with a dataset comprising 48000 time series, each consisting of 2776 points, categorized into 6 groups. A vector, "`valid_periods`," indicating the start and end points of actual data for each time series, as all time series were padded to a length of 2776. After an initial analysis and visualization of the dataset, we computed statistics, including category distribution and the minimum, maximum and average length of time series for each category.

Given the dataset's normalization between 0 and 1, we refrained from additional normalization for most tests. Although attempts were made to further normalize using the Robust Scaler, but it did not yield improved results. Prior to building sequences, padding was removed from time series and the dataset was split into training and validation sets (90/10 ratio) maintaining an equal category distribution between the two sets.

Sequences were constructed with a window size of 200 and a stride of 20 for most tests. We modified the standard function to create a sequence even if the time series was smaller than the window size, adding the necessary padding. We also experimented with discarding sequences shorter than 100 points to avoid sequences with too much padding, but it did not impact the score. Additionally, a variant of the train/validation split was created, focusing on time series belonging to specific categories.

## 3 First approach

We started by implementing some simple networks similar to the ones shown during laboratory sessions such as model composed by LSTM or GRU. We made different tests on this one, modifing both the structure and the parameters of the network, to name a few we have tried both 64 and 128 units for the recurrent layer, multiple layers of Conv1D with different kernel sizes and number of filters and also adding Batch Normalization and Dropout layers.

Our best model using this approach was made of a Bidirectional LSTM with 128 units, followed by three layers of Conv1D, MaxPooling and Batch Normalization, then another Bidirectional LSTM that sends the output to a GlobalAveragePooling which output goes to the last dense output layer.
Since this was our first attempt, we then started to develop different models to try to improve our score, but eventually this was our best scoring model.

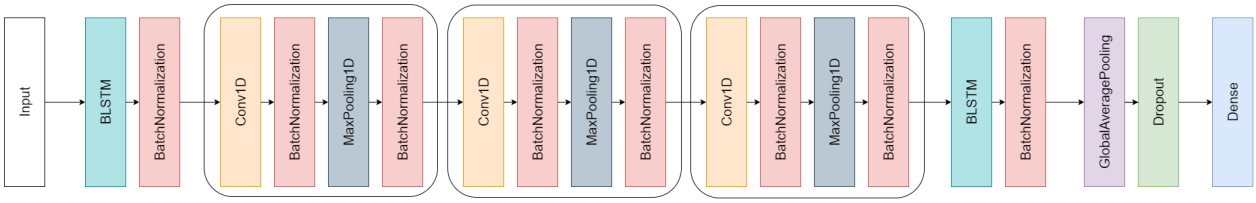



Figure 1: Architecture of our best model

## 4 Autoencoder

The first different approach we tried was an Autoencoder structure composed by an Encoder block made of 2 Conv1D layer, which each of them was connected to a Batch normalization and a Max Pooling Layer, and a Decoder block composed by a dense and 2 Up-sampling layers to re-scale the parameters. These two blocks were surrounded by two Bidirectional LSTM. Also with this architecture we tried to modify the internal structure and the parameters but none of the variants outperformed the one mentioned above.
In our tests we tried to change this model by adding an Attention mechanism [1] hoping that it would help our model gain a better understanding of the data focusing on different parts of the time series, but unfortunately this didn't improve the accuracy of the model.

## 5 Resnet-like architecture

Subsequently we approached a ResNet-like architecture composed by a Bidiridectional LSTM connect to two blocks made of 4 Conv1D, with a Dropout and a Batch normalization layer for each convolutional one, with a shortcut connection directly from the input of a block and the input of the successive one.
We trained this model with different number of blocks, various number of filter and kernel sizes for the Conv1D layers and also different numbers of Conv1D layers, but the changes on the network size were not increasing our performances. This network started to show some improvements, but it wasn't good enough to reach the top position in this challenge.

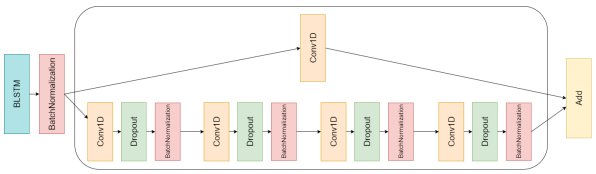



Figure 2: Scheme of the Resnet-like block we used

## 6 Ensamble model

We tried to train a model for each specific category and use the category we had in input as additional information to select which one to use for the prediction. Specifically we trained the best model mentioned in the "First Approach" section for each single category and we modified the `model.py` to select the correct model to make the prediction. Unfortunately this approach didn't have much success.

## 7 Transformer

Given the recent developments and results obtained by the Transformer architecture [1] we tried to adapt it to the time series forecasting problem. In particular we used Time2Vec [2], based on the original idea of Word2Vec, to encode the time series and then we used only the encoder part of the transformer to make the prediction for our data.
Differently from the great results obtained with semantics, transformer seems to not work so well with time series forecasting and this was further confirmed by this research [3].

## 8 Linear model

In the last days of the first phase, following the suggestion of Prof. Lomurno, that presented the topic in his paper "Two Steps Forward and One Behind" [4] and after looking at the studies in this paper [5] we tried an architecture made of linear layers to improve our results. At first we implemented a simple network made only of an input and an output layer and subsequently we developed a more complex structure by adding multiple hidden linear layers. Unfortunately we couldn't reproduce the results obtained in the two papers with our dataset.

## 9 N-Beats

After some researches, we came across this architecture [6] that outperformed other well-established models. We implemented the same type of architecture but using less blocks than the ones used in the paper because of the GPU memory constraints. In particular we used 8 blocks with 4 FC layers each, made of 512 neurons.
Unfortunately also this architecture didn't work well with our data and didn't outperform the LSTM architecture.

## 10 Autoregression vs Direct approach

In the final days of the development phase we also tried the Autoregression approach to improve our prediction based on the implementation presented in the laboratory classes. To develop this method we had to retrain our models, in particular the one mentioned in the "First approach" section and the ResNet-like model, to predict only one point in the future and then modify our prediction function in the `model.py` file to concatenate the predictions.
Even with this approach we couldn't be able to outperform our best result so we abandoned it.

## 11 Time2Vec and Attention

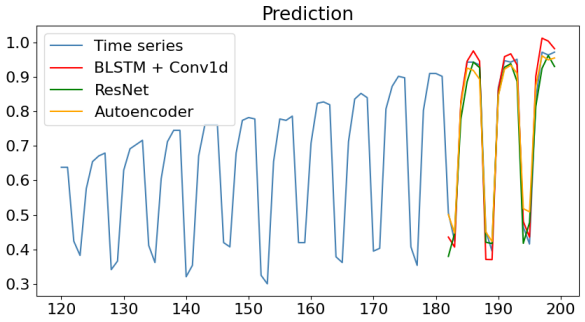
Further tests were made using all the models mentioned above, implementing both the attention mechanism and the Time2Vec encoding to try improve even more the performances, but neither of these two mechanism helped our models to perform better.

## 12 Final Considerations

We found this challenge very different from the first because we couldn't be able to find what we were missing to perform well differently, from the first one and we were a bit demoralized because of all the time spent researching new architectures or new techniques that still were not enough to achieve a score that could satisfy us.
In the end we are still happy that we could implement what we have seen and studied during the lessons and broaden our knowledge in this topic.

In this final table are presented our three best models submitted in both phases with the *Val MSE* and *Test MSE* values for each of them, with a comparison of their prediction on one of the time series in the dataset:

| **Network** | **Val MSE** | **Test MSE** |
|---|---|---|
| BLSTM + 3 Conv1D + BLSTM | 0.0.008520 | 0.01018455 |
| BLSTM + AutoEncoder + BLSTM | 0.00637854 | 0.01058138 |
| ResNet 2 blocks 4 Conv1D | 0.00843970 | 0.01069852 |



## Contributions

We equally contributed to the search and development of new models and techniques both explained during the lectures and never seen ones. Most importantly everyone contributed for the training with all the resource available and accounts divided between Google Colab and Kaggle.

## References


[1] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[2] Seyed Mehran Kazemi et al. *Time2Vec: Learning a Vector Representation of Time*. 2019. arXiv: 1907.05321 [cs.LG].

[3] Ailing Zeng et al. *Are Transformers Effective for Time Series Forecasting?* 2022. arXiv: 2205.13504 [cs.AI].

[4] Riccardo Ughi, Eugenio Lomurno, and Matteo Matteucci. *Two Steps Forward and One Behind: Rethinking Time Series Forecasting with Deep Learning*. 2023. arXiv: 2304.04553 [cs.LG].

[5] Anastasia Borovykh, Cornelis W. Oosterlee, and Sander M. Bohte. *Generalisation in fully-connected neural networks for time series forecasting*. 2019. arXiv: 1902.05312 [stat.ML].

[6] Boris N. Oreshkin et al. *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting*. 2020. arXiv: 1905.10437 [cs.LG].

[7] Dave Cote. *Hands-On Advanced Deep Learning Time Series Forecasting with Tensors*. https://medium.com/@dave.cote.msc/hands-on-advanced-deep-learning-time-series-forecasting-with-tensors-7facae522f18. 2022.

[8] Daniel Bourke. *Time series forecasting in TensorFlow*. https://github.com/mrdbourke/tensorflow-deep-learning/blob/1be0f77fbb4e056694a910f403a8f61f1b650eec/10_time_series_forecasting_in_tensorflow.ipynb. 2021.